Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors

Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, Edmond Boyer

Vision, Video, and Graphics (2003)
P. Hall, P. Willis (Editors)

Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors

JM. Hasenfratz[1] and M. Lapierre[1] and J.-D. Gascuel[1] and E. Boyer[2]

[1] Artis GRAVIR/IMAG, a joint project of CNRS, INPG, INRIA, UJF.
[2] MOVI GRAVIR/IMAG, a joint project of CNRS, INPG, INRIA, UJF.
http://www-artis.imag.fr/CYBER

Abstract

In this paper, we show how to capture an actor with no intrusive trackers and without any special environment like blue set, how to estimate its 3D-geometry and how to insert this geometry into a virtual world in ***real-time****. We use several cameras in conjunction with background subtraction to produce silhouettes of the actor as observed from the different camera viewpoints. These silhouettes allow the 3D-geometry of the actor to be estimated by a voxel based method. This geometry is rendered with a marching cube algorithm and inserted into a virtual world. Shadows of the actor corresponding to virtual lights are then added and interactions with objects of the virtual world are proposed. The main originality of this paper is to propose a complete pipeline that can computes up to 30 frames per second.*
Since the rapidity of the process depends mainly on its slowest step, we present here all these steps. For each of them, we present and discuss the solution that is used. Some of them are new solutions, as the 3D shape estimation which is achieved using graphics hardware. Results are presented and discussed.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation – Bitmap and framebuffer operations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual reality I.4.1 [Image Processing and Computer Vision]: Scene Analysis – Tracking

1. Introduction

The CYBER project† inserts a live actor into a virtual word in real-time. To achieve this operation with realism, the animator must be integrated perfectly in its virtual environment. This perfect insertion requires more than a simple composite of the video flow over the rendered 3D-scene. First, we must add shadows of the actor as would be casted by the lights in the virtual scene. This prevents the impression that the actor flies above the floor. We must also relight the video flow to take account of the virtual environment lighting. Without that, combined pictures are not convincing and a virtual red light near the actor does not impact its visual aspect. Finally, just combining 3D environment and video data does not allow any interaction with the virtual scene.

This topic is very important for television industry using virtual sets and on-line presentations, for games where the player is rendered into the game, for education where the student visits a virtual place.

In this paper, our objective is to demonstrate that it is feasible to capture an actor with no intrusive trackers and without any special environment or material, estimate shape, and combine it with a virtual world up to 30 times per second.

The paper is organized as follows: previous work are reviewed in section 2. Section 3 describes the hardware and software architecture which are used. Section 4 is devoted to the capture stage; in particular camera calibration and background subtraction are discussed. Section 5 presents our original approach to real-time shape estimation which is based on graphics hardware. Section 6 proposes two examples of actor integration into a virtual world: high quality real-time shadows of the actor and real-time interaction with

† http://www-artis.imag.fr/CYBER

virtual objects in the scene. Finally we discuss the results before concluding.

2. Previous Work

Several approaches have been proposed to insert a live actor into a virtual scene at interactive rates. 3D information can be retrieved using stereo cameras which generate range images[4, 7]. This technique is able to reconstruct concave regions and the Virtualized Reality system[24] shows that it can work on large dynamic objects. Matusik[14] computes a visual hull from multiple camera views, using epipolar geometry, and generates a 3D textured model (each camera retrieving a colored silhouette).

Geometry information can also be retrieved by *Voxel Coloring*[17]: it consists in testing color consistency of voxels from the surface of the geometry among all the points of view.

Another approach is to use voxel carving: a volume is discretized into voxels, which can be used as a step to fit a skeleton[20], or to analyze human movements by fitting ellipsoids[1].

An interesting application by Lok[11] is to combine advanced interactions within the virtual world.

These methods have some drawbacks: concave regions which are difficult to capture or the number of views which are needed, but combining them as in Li[10] can reduce these problems and can improve the speed as well as the quality of the estimated geometry.

All these methods are working at interactive rates, but not at 30Hz on full VGA resolution images and with standard cameras. The solution that we propose does not have such limitation, and we will describe how to maintain this frame rate all along the process.

3. System architecture

3.1. Hardware architecture

Our hardware architecture is described in Figure 1. It is composed of four standard IEEE 1394 cameras (Sony DFW–VL500) running at a resolution of 640x480 pixels. The color mode which is used is YUV4:2:2. Each camera is connected to a Pentium4 PC running at 1.6 GHz. We use a 1Gbit/sec Ethernet network between the PCs and a SGI Onyx 3000–IR3. The Onyx is configured to use 8 R12000 processors. The output of the Onyx is a 100Hz display screen.

The actor can move inside a 2m large square. We do not use blue background (see Figure 2) but we use a controlled lighting environment with a grid of fluorescent lights. The cameras are not externally synchronized.

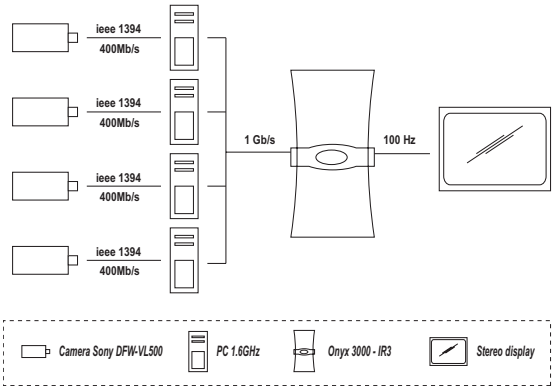

Figure 1: *Hardware configuration of the system.*

Figure 2: *The experimental setup. Cameras are surrounded by red circles in the image.*

3.2. Software architecture

PCs are running Windows 2000 with an IEEE 1394 library developed at Carnegie Mellon University[23] to drive the cameras. The OpenCV[15] library is used for camera calibration.

The different modules are shown in Figure 3. Cameras are set in continuous mode and produce 30 frames per second. On each PC, an infinite loop process subtracts the background (see Section 4.2) and filters the images. Results are sent to the Onyx via the network. Background subtraction and filtering takes less than 33ms per frame, and thus a data flow of 30Hz can be achieved. This rate is perfectly constant as the whole process if waiting for the camera capture, and this guarantees coherency between cameras (gap between two cameras is less than 1/30 second). Working with an external trigger would have lowered the frame rate to 15Hz.

Five processes run in parallel on the Onyx. Four of them are just buffering images coming from the *subtraction/filtering* modules (via the network). The fifth process is an infinite loop taking alternatively data from each buffer and therefore synchronizing the flows coming from the cameras. This last process estimates the 3D geometry of the actor

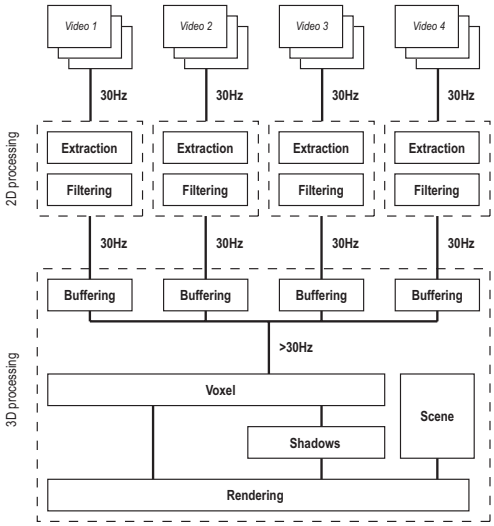

Figure 3: *The different modules involved (dashed rectangles correspond to processes.)*

(see Section 5) and performs the integration with the virtual world (see Section 6).

4. Capture

4.1. Camera calibration

Calibration is required to recover positions, orientations and internal parameters of the cameras involved in the reconstruction process. These information are needed to project voxels into image planes and verify which of them belong to the silhouette cones and therefore to the estimated shape. The calibration information are encoded in projection matrices, one per image, which are estimated in a preliminary step. Note that we therefore assume that all the camera parameters remain constant during the real time 3D environment and video data combining step. In order to estimate the projection matrices, we use a calibration pattern with known Euclidean characteristics: a chess-board. Since exact Euclidean characteristics of the scene are not needed for combining virtual and real images but only its geometry up to a scale, we do not use the exact Euclidean dimensions of the chess-board's squares but just the fact that they are of equal dimensions. Estimating projection matrices from coplanar points with known positions is then a well known problem, see [22, 26] for instance. We use the *Open Computer Vision Library*[15]. Such library offers solutions for both corner extraction and projection matrix estimation. Note that camera distortion could also be taken into account using the OpenCV library. However, such distortion does not appear to be significant in our case and we neglect it. Finally, we mention that we are currently implementing a new calibration system based on a more flexible pattern than the chess-board which is difficult to position efficiently in the scene.

4.2. Background subtraction

In our application, relevant and irrelevant information in the images correspond to foreground and background information respectively, where the background is assumed to be static and can therefore be learned *a priori*. Thus, extracting relevant information -the silhouettes- consists in detecting foreground, or equivalently background, pixels in the images. The solution generally adopted for that purpose is to verify pixels' values and detect changes. Indeed a background pixel value should be constant over the time while foreground pixel value should vary at some time. Following this principle, several approaches exist that check the intensity functions at each pixel. Robust ones use temporal filters to detect changes as well as spatial filters to cluster pixels and eliminate false detection (see [21] for a comparative study). However, in our context the critical issue is not robustness but how fast the operation is. Indeed, even if artifacts exist, they are usually not consistent over the image set and thus removed during the reconstruction step. Consequently, we use a simple but fast method. Background pixels are assumed to follow Gaussian distributions, possibly correlated, in the YUV space. Such distributions are learned *a priori* from a set of background images. The fact that a pixel belongs to the silhouette or the background is then checked by thresholding its Mahalanobis distance to the background mean position in the YUV space. Note that in order to take into account shadows during the subtraction, constraints on the intensity (the Y parameter) values could easily be relaxed. As a result, we obtain a flow of black and white pictures representing the silhouettes as viewed by the different cameras (see Figure 4). This operation takes in average 22ms per image.

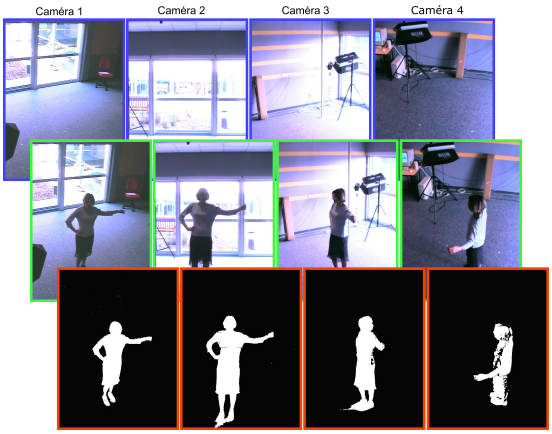

Figure 4: *Background subtraction: the first row shows background images, the second row some images taken during runtime, and the third row displays the corresponding extracted silhouette bitmaps.*

We have also implemented and tested binary morpholog-

ical filters to remove artifacts such as isolated pixels. However, the benefit is not really important while time consuming is significant. Finally, the silhouette extraction takes less than 23 ms per frame. The resulting bitmaps are sent to the buffering modules via the local network. The flow is cadenced by internal cameras clocks which deliver images at 30Hz. The output of the buffering modules is then used for shape estimation as explained in the next section.

5. Shape estimation

The next step is to estimate the scene geometry in order to compute illumination interactions. We are given, at each instant, several silhouettes corresponding to different camera viewpoints. The shape that can be estimated from these silhouettes is the visual hull [8] of the objects under consideration. The visual hull is in fact the maximal solid shape consistent with the object silhouettes. Such an approximation of the scene captures all the geometric information available from the silhouettes. Several approaches have been proposed to compute visual hulls. They can be roughly separated into two categories: volume based approaches and surface based approaches.

The first category includes methods that approximate visual hulls by collections of elementary cells called voxels and carve them according the silhouette information (See [18] and [5] for reviews). Approaches in this category can handle objects with complex topologies. Note that due to the space discretizations, they lead to approximations only of the visual hull, and that such approximations can be computationally expensive when precision is required.

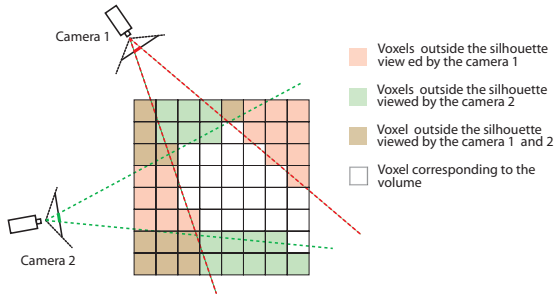

Figure 5: *Principle of voxel carving.*

The second category of approaches estimates elements of the visual hull's surface by intersecting the viewing cones associated the silhouettes [2, 13]. These category of approaches suffer from numerical instabilities [9]. Consequently, they often lead to surface models which are incomplete or corrupted, in particular when considering objects with complex topologies. Also, the visual hull itself is an approximation of the scene geometry and an exact description of an approximation of the scene is not necessarily required to compute illumination interaction. Thus, we have chosen for our application a voxel-carving approach (see Figure 5).

The accuracy of reconstructed geometry is depending on the way cameras are situated in the scene: in our case we wanted to be able to reconstruct a whole body, so the reconstructed space is $2m^3$ large. At a 64^3 resolution each voxel in this case is 3cm large. We could work on smaller objects with a better precision by focusing cameras on a reduced space, for example to reconstruct only the hands of the actor with a 0.5cm accuracy.

The following part describes how graphics hardware can be efficiently used for a voxel-carving implementation.

5.1. Hardware assisted voxelisation

Classically, a cube of N^3 voxels is considered as the discrete space to be carved. In this paper, we also use N^3 cells but instead of voxels, we consider N images of N^2 pixels. These images are located in the middle of each voxel slices. With this approach, we can benefit from graphic hardware at different stages of the process as shall be seen. In this section, we consider N=64 pictures of N by N pixels. A discussion of the parameter N is proposed in Section 7.

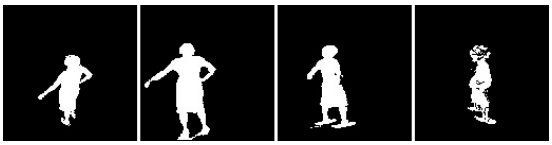

Figure 6: *Four textures corresponding to the silhouettes coming from the different buffer modules.*

We read one silhouette in each buffer and transform it into an OpenGL texture of 128x128 pixels as in Figure 6.

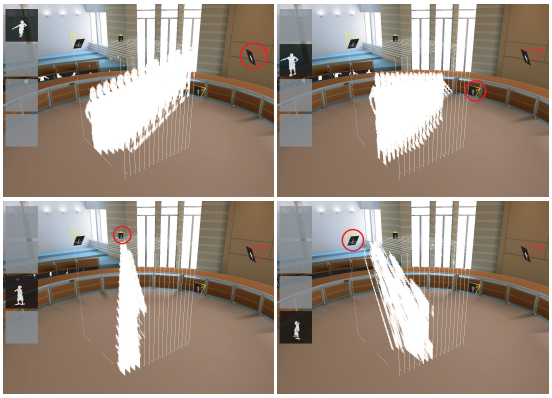

Figure 7: *Projections of the silhouettes onto the different slices for each camera. Red circles identify the involved cameras in each image.*

For each camera, we project the silhouette textures in voxel-space using graphics hardware facilities. This approach is similar to the one used by Lok[11] but with fewer

constraints. Figure 7 shows the depth projections on 64 planes corresponding to the grid of 64^3 voxels. Note that to ensure correct texture projections, the glHint parameter in OpenGL must be set to the highest level.

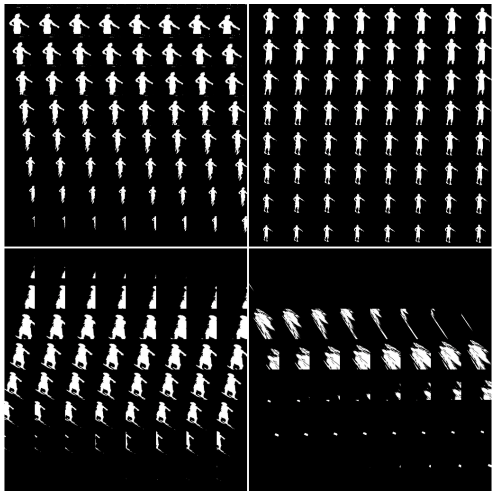

Figure 8: *Texture projection tiled in a single 512x512 texture map (8x8 textures of 64x64 pixels).*

All the projected textures corresponding to the same camera viewpoint are tiled in a single texture map of 512x512 pixels. This size corresponds to 64 projections of the silhouette textures composed by 64x64 pixels (see Figure 8).

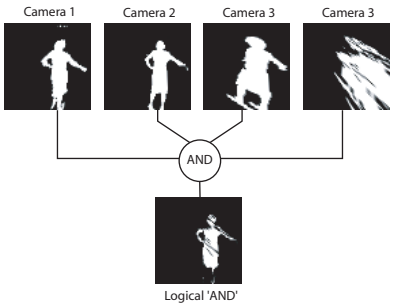

Figure 9: *Logical* AND *operation on four textures projected to the same slice.*

To determine the volume corresponding to the scene visual hull we must find the intersection between the projection of all bitmap images. Again, we take here advantage of the hardware: blending facilities of OpenGL can simulate the AND operation that is used for the intersection: by calling `glBlendFunc(GL_DST_COLOR, GL_ZERO)`, successive renderings of projected silhouettes will be combined with previous ones using the AND operator.

Figure 9 shows the result of the *AND* logical operation with the different projected texture on the same plane. Note that we always work with the same planes representing the slices of the original voxel grid.

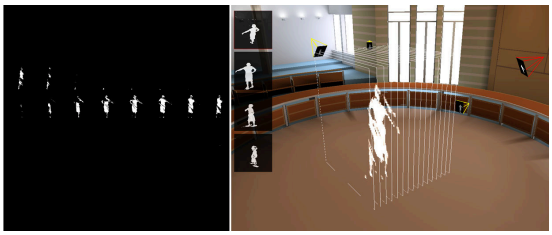

Figure 10: *Logical* AND *operation on four tiled textures. On the left, the tiled texture is shown, on the right all the textured quads located in the virtual world are displayed.*

Figure 10 presents the result of the *AND* logical operation on the entire texture and its equivalent operation in the voxel-space.

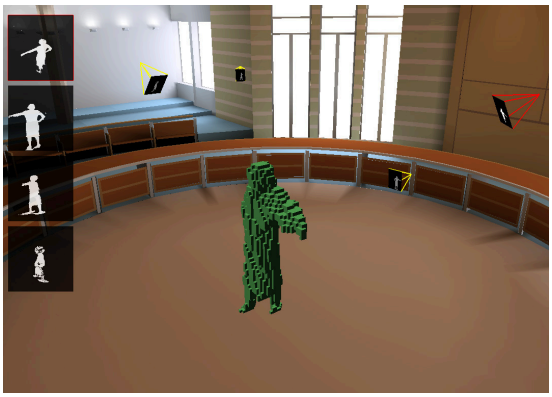

Figure 11: *Voxel representation of the* AND *operation on four tiled textures.*

Finally, to obtain the voxels representing the actor we just consider each white pixel in the final tiled texture map as a voxel (see Figure 11).

5.2. Pseudocode for the reconstruction

The different steps of the shape estimation algorithm are as follow:

```
Disable depth test
Disable blend
For each camera
  Transform silhouette to texture
  Set texture matrix to camera parameters
  For each slice
    Set working area to a tile
    Render a quad with texture projected
  Enable blend
Read whole frame buffer (glReadPixels)
Convert white pixels to voxels
```

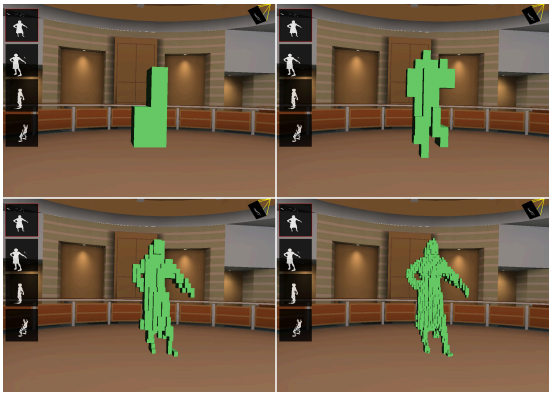

Figure 12: *Different resolutions of the voxel grid (8^3, 16^3, 32^3, 64^3).*

5.3. Marching cube on binary voxels

To produce a triangulated surface from the voxel data, we use the classical marching cube[12] algorithm. From an implicit function defined over space (the voxels reconstructed so far), the goal is to derive a smoother surface by reconstructing triangle strips (see figure 13).

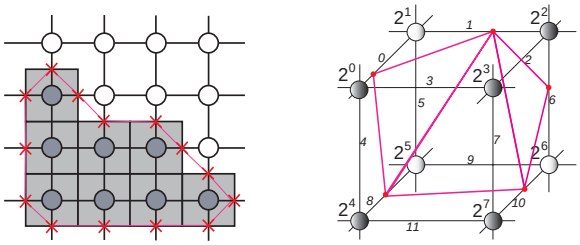

Figure 13: *Left: the relation between the input voxels (shaded squares), the in or out values on each grid corner, and the reconstructed contour. Right: in 3D, a table is used to list triangle strips. The code of the cell, computed from the corner values, is used to index a fixed table of edge lists. In the case shown (`10011101` = 157), we have `strips[157] = {6, 10, 1, 8, 0, END}`.*

The implementation used provides a slight improvement over the original algorithm for *OpenGL* rendering, because we pre-compute stripping into the *strips* table. A simple combinator pass over the table combines triangles sharing one edge into strips, and reduces the total number of vertices by 29% (and the data send by only 21%, because of the `STOP` code(s) inserted when more than one strip is needed for a cell). But in practice, a speedup of approximatively 2 is observed, because the triangle do share edges, but also because all the cells don't have the same compression ratio, nor the same probability.

The final result is shown in figure 14. The roughness observed (and particularly the normal quantization) is due to the binary nature of the input data. A smoothing should be used at that level to improve the surface regularity.

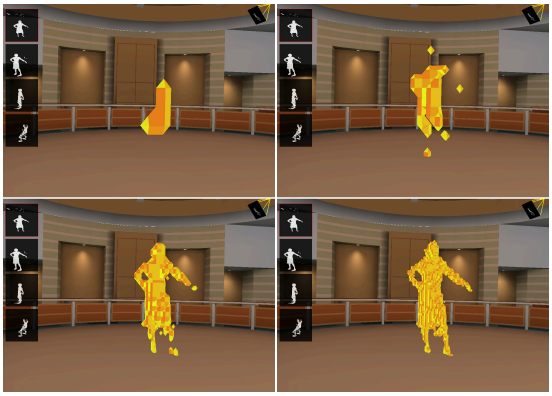

Figure 14: *Different resolutions of the marching cube (8^3, 16^3, 32^3, 64^3).*

6. 3D Graphics integration

The 3D model can be used for visual purposes: it can be rendered "as is" in a virtual world with points or cubes (see Figures 12 and 14), but it also permits volumetric effects and interaction.

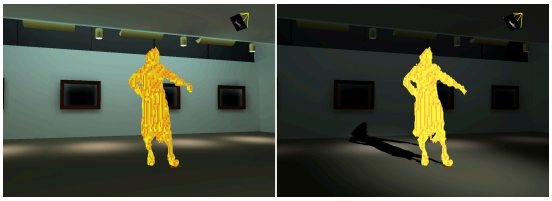

Figure 15: *Marching cube representation: Shadow due to a virtual light. Left: the actor seems to be "flying" above the floor. Right: shadows remove this impression*

To integrate the avatar in a realistic way in a virtual scene we must add cast shadows. These shadows help understanding relative object position in the virtual scene. In particular, it prevents the actor from "flying above" the floor (see Figures 15). Many real-time techniques, based on geometry[3, 6] or on a bitmap representation[25, 16] can be used. Since the geometry is changing at each frame, we chose a simple projective shadow technique which is fast, does not depend on geometry complexity and cast shadows on any geometry. Shadows in the scene are precalculated and we only need to calculate at each frame the shadows cast by the avatar. This is done by rendering a black and white texture representing the avatar as seen from the light source and by projecting this image on the scene geometry following the light direction.

Another advantage to dispose of 3D information is that

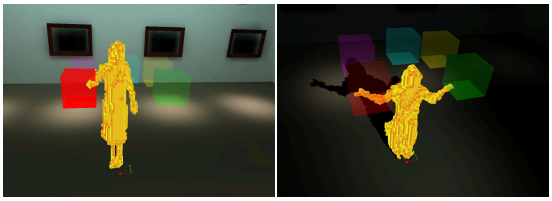

Figure 16: *Example of 3D interaction. The actor could interact with the five virtual cubes.*

we can calculate collision between the avatar and its virtual environment. We implemented a virtual synthesizer, symbolized by five semi-transparent cubes that play different sounds when the avatar collides them (see Figure 16). Another example implemented was interaction with a virtual light, with a 3D interface (same cubes) allowing a light source to move around the avatar. Note that we don't match a skeleton to the avatar (like Theobalt[20]), so we can't detect which part of the body is colliding. We only used the percentage of filled voxel in different particular regions of our grid of voxel.

7. Results

As seen in section 4.2, it takes less than 25ms to capture and extract the actor, that permits taking advantage of the maximum performance from cameras. On the server part, we show in detail how we reconstruct also at 30fps.

The following table shows precisely how much time is spent in each processing step. The reconstruction time is the most important. It is distributed between the number of renders done (N per camera, where N is the resolution of voxel space) and the transfer from frame buffer (N^3 pixels) to our voxel data structure in main memory. As each of this call is time consuming, we do it only once by previously rendering the whole N^3 pixels.

step	*time*	*total*
scene rendering (34.000 triangles)	6ms	6ms
+ receiving 4 textures	2ms	8ms
+ reconstruction 64^3	15ms	23ms
+ rendering as cubes	3ms	26ms

Time for each step, with 4 cameras and 64^3 *voxels*

We integrate our reconstruction into several virtual scenes with different resolutions to see the limits of our method. The following table show results obtained in a 34.000 polygons scene (see Figure 12). We can see that the 30fps constraint is respected even at the highest voxel resolution.

voxel-space resolution	*time(fps)*	*with shadow*
64^3	26ms(33Hz)	41ms(20Hz)
32^3	13ms(50Hz)	18ms(50Hz)
16^3	10ms(50Hz)	14ms(50Hz)

Reconstruction and rendering performance

8. Conclusion

As demonstrated in section 7, we have been able to achieve the real-time constraint all over the pipelined processing. We should emphasis that it was a tough work to spot out the bottlenecks, and to solve them. In particular, we are running at the maximum camera's rate and definition, and at the maximum of the texture size allowed on the *Onyx's IR3*. So this is the best we could possibly obtain from that setup.

To go even further, we plan to upgrade the setup in the following ways:

- By connecting a video camera to the Onyx server, we will render a color image mapped onto the geometry. As 3D geometry of the avatar is known, we can relight each pixel of this image according to the virtual lighting.
- Add more contour cameras (e.g. 2 or 4 by PCs) to have tighter volume carving, and less ambiguities. The next setup targeted will start with 20 cameras.
- Use higher definition contour cameras (e.g. *Sony VL900*) running at 2048*x*2048 @ 10Hz. And use time multiplexing to keep a good frame rate.
- Use several graphic cards to compute the voxel carving, and improve geometry extraction by running the marching cube on a continuous *inside-ness* function.

Furthermore, video integration into the virtual scene could be improved by using real-time soft smooth shadows, for instance inspired from [19].

We're also working on interacting with the virtual worlds: because we have a real-time reconstruction of the actor's shape, we are able to compute contacts between the real actor and the virtual set. This open us the exciting world of live virtual experiences.

Acknowledgments

Thanks to Isabelle, who gracefully accepted to dance in front of our cameras. This work was supported by the "ACI Jeunes Chercheurs" of the Department of the Research.

References

1. Kong Man Cheung, Takeo Kanade, J.-Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *Proceedings of the*

2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '00), volume 2, pages 714 – 720, June 2000.

2. R. Cipolla and P.J. Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 1999.
3. Franklin C. Crow. Shadow algorithms for computer graphics. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 242–248. ACM Press, 1977.
4. Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM Press, 1996.
5. C.R. Dyer. Volumetric Scene Reconstruction from Multiple Views. In L.S. Davis, editor, *Foundations of Image Understanding*, pages 469–489. Kluwer, Boston, 2001.
6. Heidmann. Real shadows, real time. In *Iris Universe*, pages 23–31. Silicon Graphics Inc., 1991.
7. Sing Bing Kang and Richard Szeliski. 3-d scene data recovery using omnidirectional multibaseline stereo. Technical Report CRL 85/6, Cambridge Research Lab, 1996.
8. A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *pami*, 16(2):150–162, February 1994.
9. S. Lazebnik, E. Boyer, and J. Ponce. On How to Compute Exact Visual Hulls of Object Bounded by Smooth Surfaces. In *cvpr01*, volume I, pages 156–161, December 2001.
10. Ming Li, Hartmut Schirmacher, Marcus Magnor, and Hans-Peter Seidel. Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *Proc. 5th Conf. on Multimedia Signal Processing*, 2002.
11. Benjamin Lok. Online model reconstruction for interactive virtual environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 69–72. ACM Press, 2001.
12. William E. Lorensen and Harvey E. Cline. Marchnig cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.
13. W. Matusik, C. Buehler, and L. McMillan. Polyhedral Visual Hulls for Real-Time Rendering. In *Eurographics Workshop on Rendering*, pages 115–126, 2001.
14. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.
15. Open computer vision library. http://sourceforge.net/projects/opencvlibrary/.
16. Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252. ACM Press, 1992.
17. Steven Seitz and Charles Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. *International Journal of Computer Vision*, 25(3), November 1999.
18. G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafe. A survey o methods for volumetric scene reconstruction from photographs. In *International Workshop on Volume Graphics*, 2001.
19. Cyril Soler and François Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics Proceedings*, pages 321–332, Jul 1998. Annual Conference Series, SIGGRAPH'98.
20. C. Theobalt, M. Magnor, P. Schüler, and H.-P. Seidel. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. *Proc. IEEE Pacific Graphics 2002), Beijing, China*, pages 96–103, oct 2002.
21. K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and Practice of Background Maintenance. In *iccv99*, pages 255–261, 1999.
22. Roger Y. Tsai. An Efficient and Accurate Camera Calibration Technique for 3d Machine Vision. *Proceedings of IEEE Coonference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.
23. Iwan Ulrich, Christopher Baker, Bart Nabbe, and Illah Nourbakhsh. Ieee-1394 digital camera windows driver. http://www-2.cs.cmu.edu/ iwan/1394/.
24. Sundar Vedula, Peter Rander, Hideo Saito, and Takeo Kanade. Modeling, combining, and rendering dynamic real-world events from image sequences. In *Proc. 4th Conference on Virtual Systems and Multimedia (VSMM98)*, pages 326 – 332, November 1998.
25. Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.
26. Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 22(11):1330–1334, 2000.